TITLE: IRIS DATA ANALYSIS

AUTHOR: FALAK ZOYA KHAN

Introduction: The Iris dataset is used to classify flowers into three species based on their features, including sepal length, sepal width, petal length, and petal width.

Objective: To explore and analyze the Iris dataset, build machine learning models, and accurately classify iris species.

Data Exploration: The dataset is explored to understand its structure, check for missing values, and gain insights into the features.

Data Preprocessing: Necessary steps include encoding categorical variables, normalizing or scaling data, and splitting the dataset into training and test sets.

Model Building: Logistic RegressioN and SVM classification models are trained to classify the iris species.

Results: The best-performing model is selected, and conclusions are drawn about the accuracy and efficiency of different models.

Tools Used: Python libraries like pandas, matplotlib, scikit-learn, and seaborn are used for data manipulation, visualization, and model building.

IMPORT LIBRARIES

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

STEP 1: LOAD THE DATASET

In [ ]:
```
# Load the Dataset:
df = sns.load_dataset('iris')
df.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

STEP 2: UNDERSRTAND THE DATA STRUCTURE

In [ ]:
```
# Check no of rows:
rows = df.shape[0]
print("no of rows: ",rows)


# Check no of columns:
col = df.shape[1]
print("nop of columns: ",col)

# Check columns name:
df.columns
```

```
no of rows:  150
nop of columns:  5
```

Out[ ]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

IDENTIFY MISSING VALUE

In [ ]:
```
# Check missing values:
df.isnull().sum()
```

Out[ ]:

	0
sepal_length	0
sepal_width	0
petal_length	0
petal_width	0
species	0

dtype: int64

DROP DUPLICATES

In [ ]:
```
# Total rows before removing shape:
rows_before = df.shape[0]

# Remove duplicates:
df= df.drop_duplicates()

# Total rows after duplicates:
rows_after = df.shape[0]

print("rows before : ",rows_before)
print("rows_after: ",rows_after)
```

```
rows before :  150
rows_after:  149
```

TRIM WHITESPACE IN CELLS

In [ ]:
```
# Remove space from specific column:
df["species"]=df["species"].str.strip()
df.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
# information about the data:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 149 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  149 non-null    float64
 1   sepal_width   149 non-null    float64
 2   petal_length  149 non-null    float64
 3   petal_width   149 non-null    float64
 4   species       149 non-null    object 
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

EXPLORATORY DATA ANALYSIS(EDA):

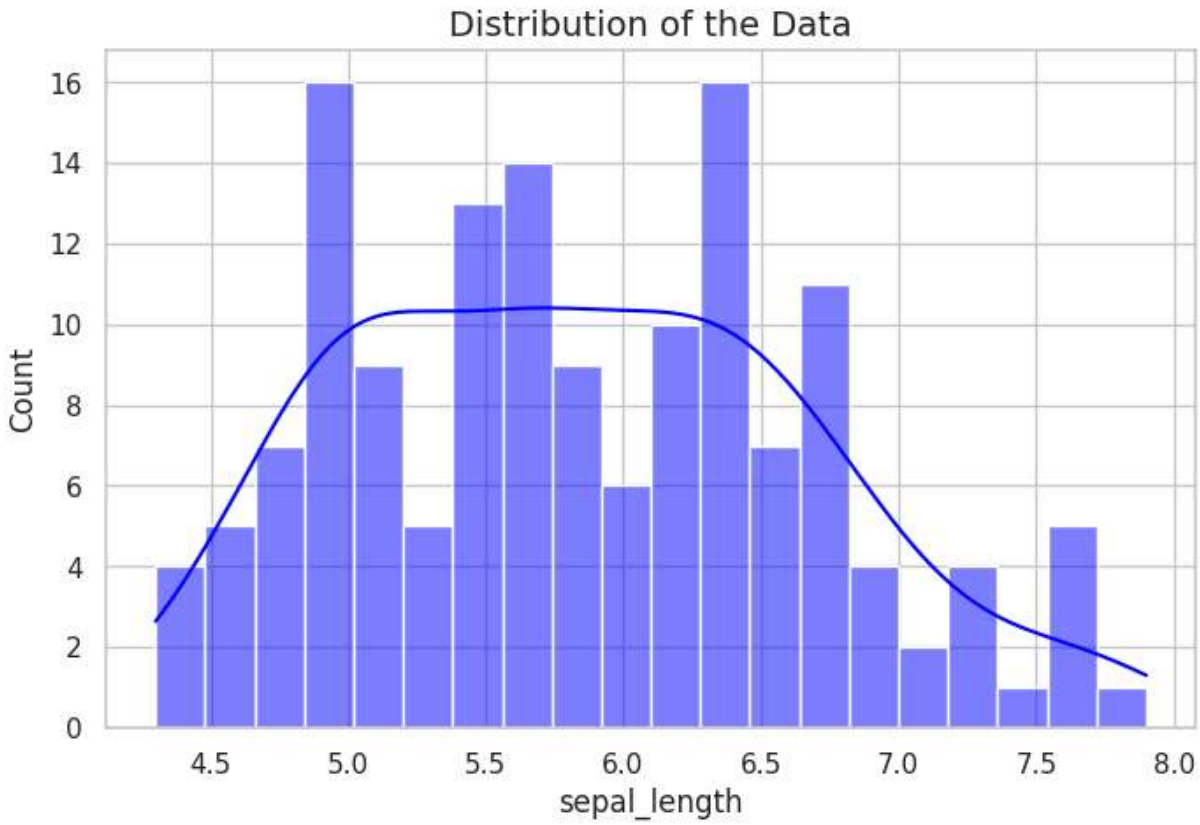
In [ ]:
```
# Summary for Statistic data:
df.describe()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width
count	149.000000	149.000000	149.000000	149.000000
mean	5.843624	3.059732	3.748993	1.194631
std	0.830851	0.436342	1.767791	0.762622
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

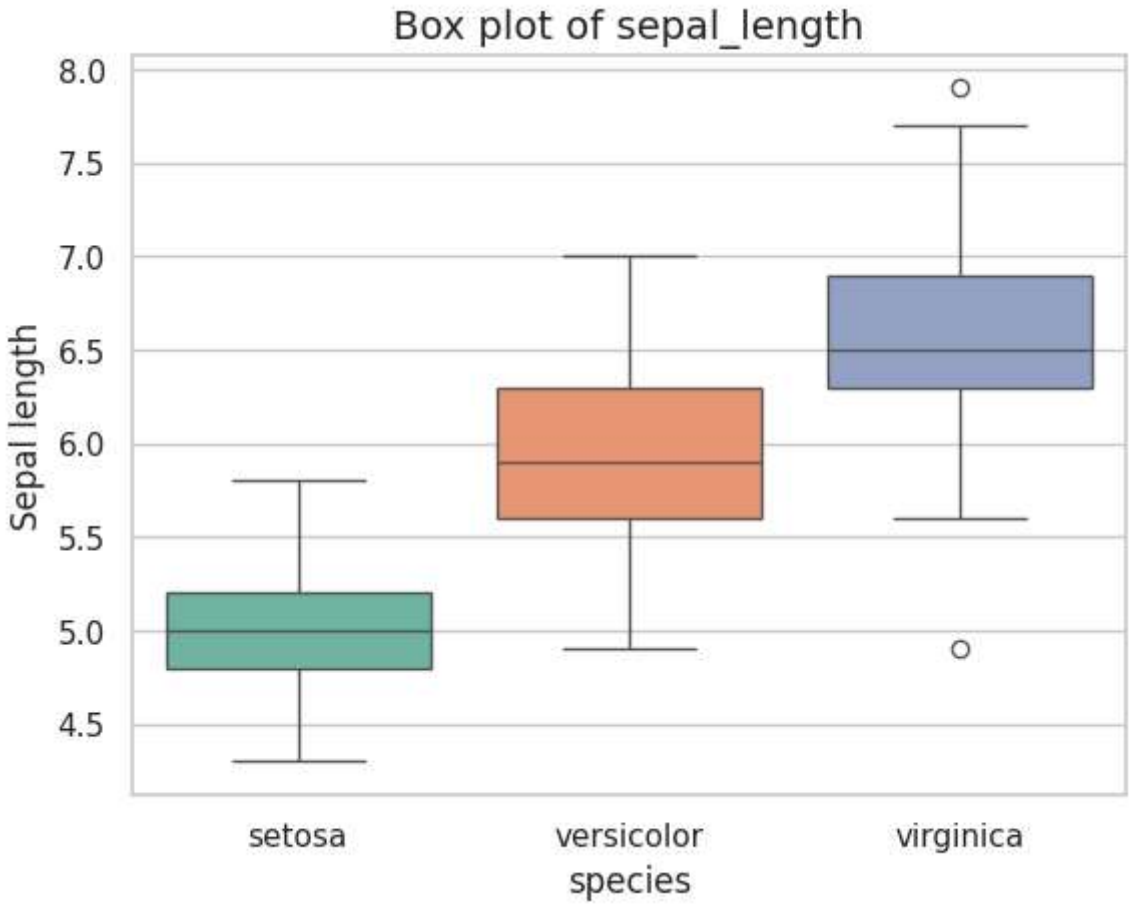
VISUALIZE THE DATA

In [ ]:
```
# Plot histogram to see distribution:
# set the style for better aesthetics
sns.set(style="whitegrid")

# plot histogram with KDE:
plt.figure(figsize=(8,5))
sns.histplot(data=df, x="sepal_length", kde=True, color="blue", bins=20 )

# Add title :
plt.title("Distribution of the Data", fontsize=14)
plt.show()
```

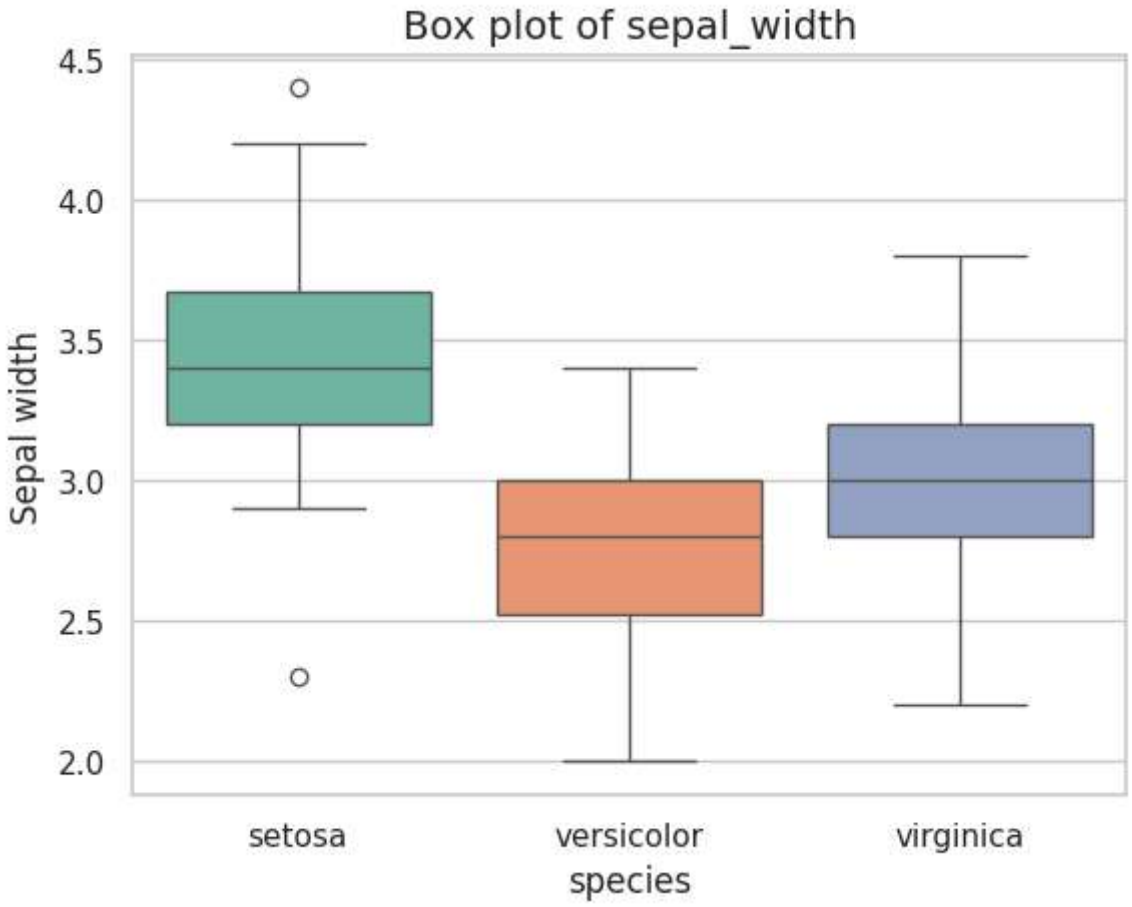


In [ ]:
```
# Plot box plot for all features:
for feature in ['sepal_length','sepal_width','petal_length','petal_width']:
   sns.boxplot(data=df, x='species', y=feature, palette='Set2')
   plt.title(f'Box plot of {feature}',fontsize=14)
   plt.xlabel("species",fontsize=12)
   plt.ylabel(feature.replace('_', ' ').capitalize(),fontsize=12)
   plt.show()
```

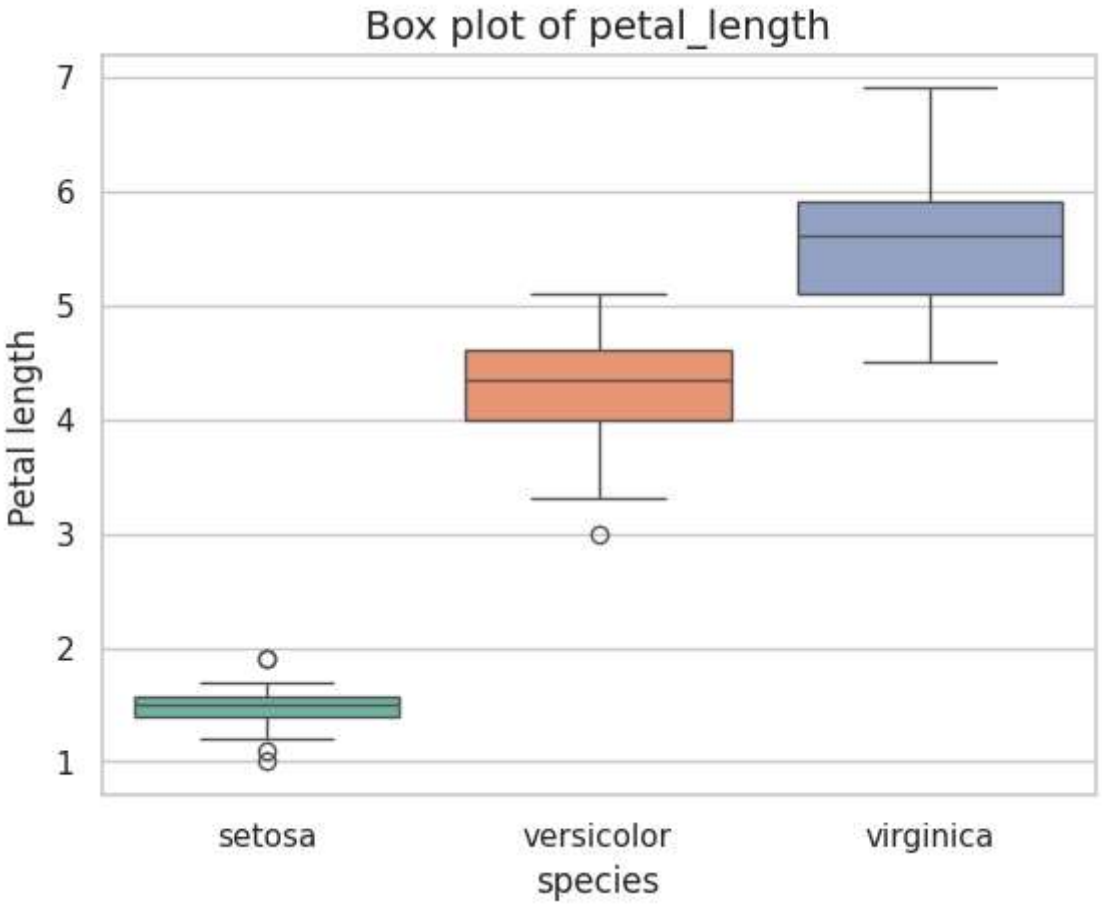
```
<ipython-input-10-7084e3a35902>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='species', y=feature, palette='Set2')
```



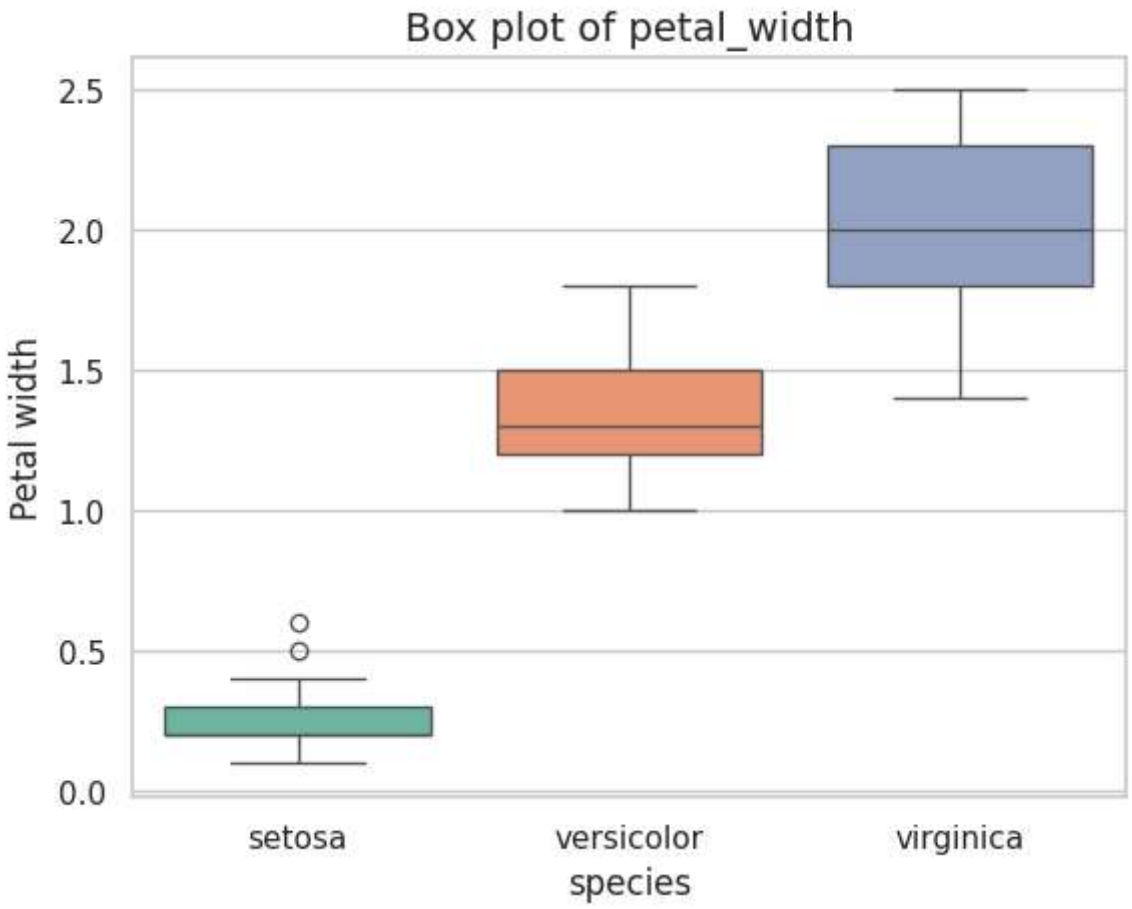

```
<ipython-input-10-7084e3a35902>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='species', y=feature, palette='Set2')
```



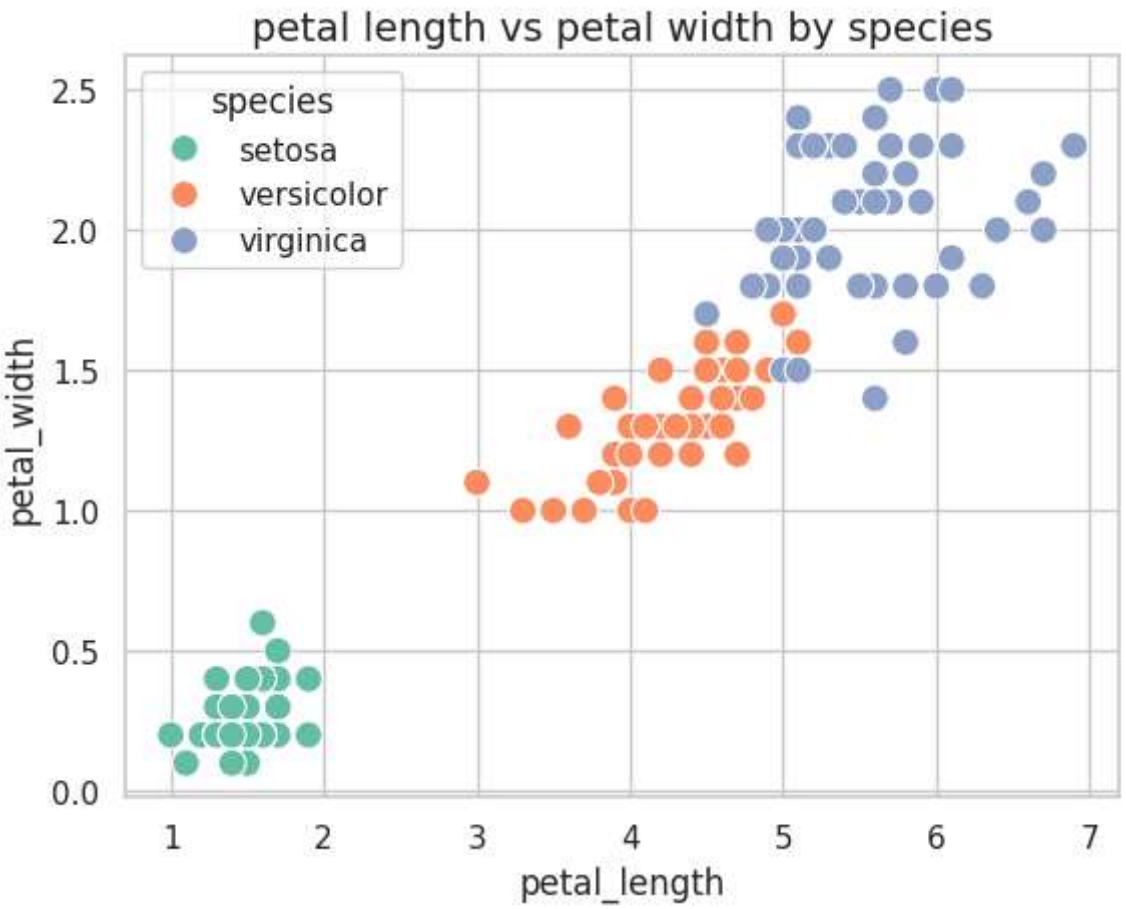

```
<ipython-input-10-7084e3a35902>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='species', y=feature, palette='Set2')
```



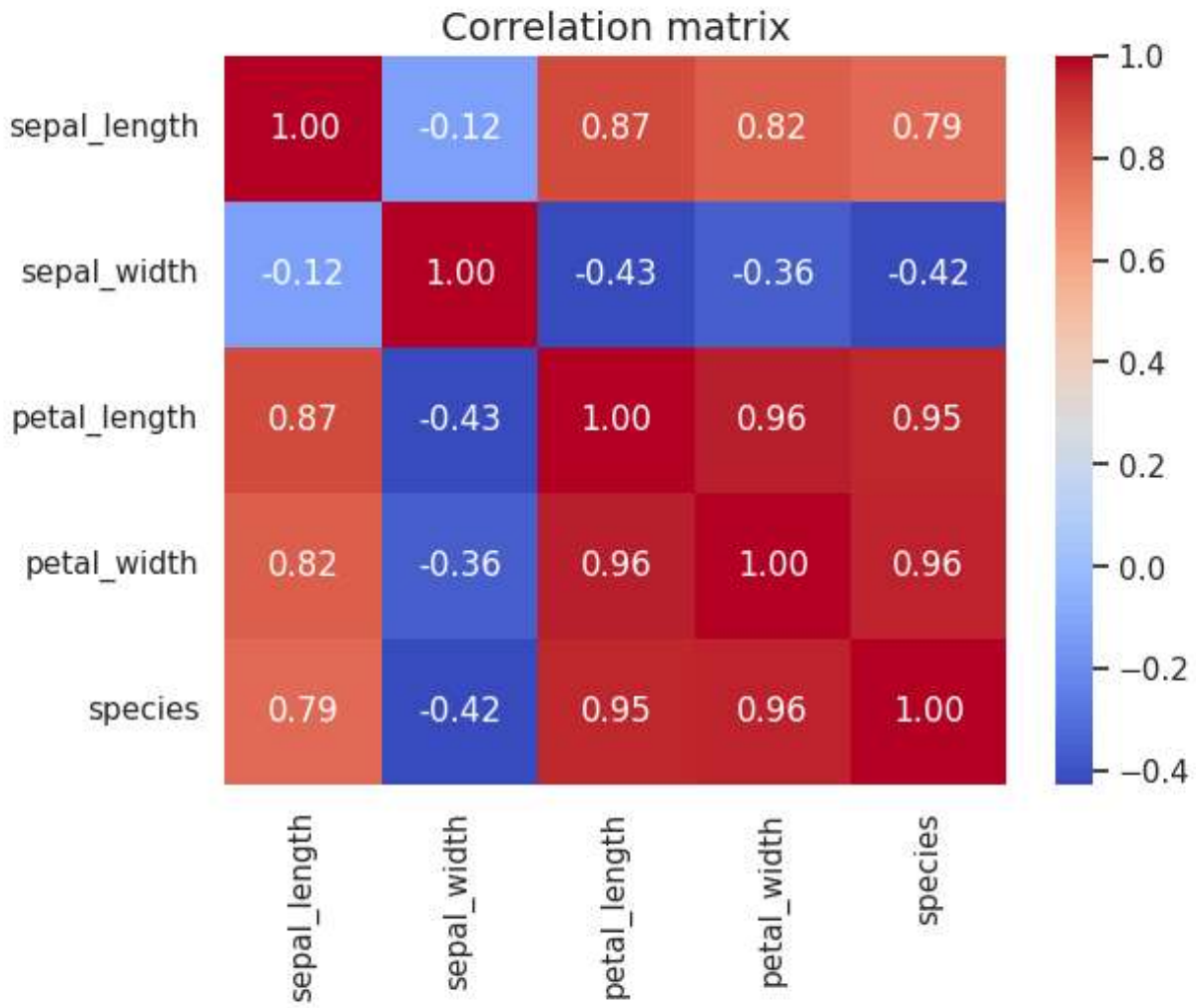

```
<ipython-input-10-7084e3a35902>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='species', y=feature, palette='Set2')
```



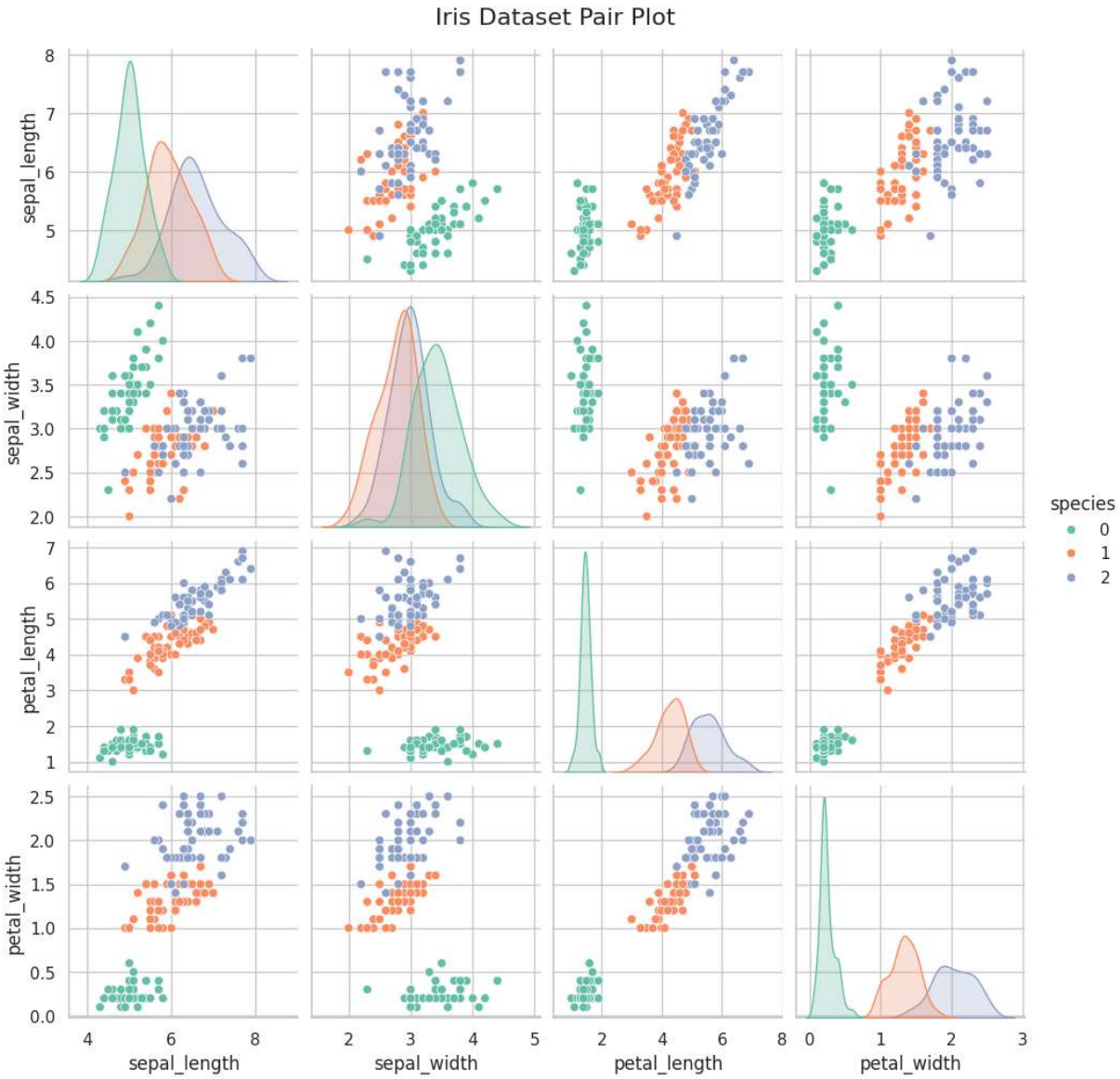
In [ ]:
```
# Scatter plot:
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species', palette=
plt.title("petal length vs petal width by species",fontsize=14)
plt.xlabel('petal_length',fontsize=12)
plt.ylabel('petal_width',fontsize=12)
plt.grid=(True)
plt.show()
```



In [ ]:
```
# convert categorical column species into numeric codes:
df["species"] = df["species"].astype("category").cat.codes

# Analyze correlation between features:
# compute correlation matrix:
corr=df.corr()

# Heatmap for correlation:
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", cbar=True)
plt.title('Correlation matrix',fontsize=14)
plt.show()
```



In [ ]:
```
# Create the Pair Plot
pair_plot = sns.pairplot(df, hue='species', palette='Set2',diag_kind='kde', height=
pair_plot.fig.suptitle("Iris Dataset Pair Plot",y=1.02, fontsize=16)
plt.show()
```



PREPARING DATA FOR MODELING

In [ ]:
```
# Step 1 :  Define Predictators and Target Variable
X = df[["sepal_length","sepal_width","petal_length","petal_width"]] # Independent v
y = df[["species"]] # Dependent variable
```

In [ ]:
```
# Step 2 : StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

PREDICTIVE MODELING

In [ ]:
```
from sklearn.model_selection import train_test_split
# Split the data:
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_stat
```

In [ ]:
```
# Print result for Verification:
print("X_train shape: ",X_train.shape)
print("X_test shape: ",X_test.shape)
print("y_train shape: ",y_train.shape)
print("y_test shape: ",y_test.shape)
```

```
X_train shape:  (119, 4)
X_test shape:  (30, 4)
y_train shape:  (119, 1)
y_test shape:  (30, 1)
```

TRAIN A LOGISTIC REGRESSION MODEL

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Ensure y in 1D:
y_train = y_train.ravel()


# Initialize and train the LogisticRegression
model = LogisticRegression(max_iter=200)
model.fit(X_train,y_train)


# Display model coefficient and intercept:
print("Model Coefficients(for all features): ",model.coef_)
print("Model Intercept(for all classes): ",model.intercept_)

# make predictions using test data
y_pred_logreg = model.predict(X_test)

# Calculate accuracy of the model
accuracy_logreg = accuracy_score(y_test,y_pred_logreg)
print(f'logistic Regression Accuracy: {accuracy_logreg * 100:.2f}%' )
```

```
Model Coefficients(for all features):  [[-0.4054145   0.94452715 -2.36685009 -0.9944
2899]
 [ 0.47040385 -0.27984798 -0.20388553 -0.74734318]
 [-0.06498935 -0.66467917  2.57073562  1.74177217]]
Model Intercept(for all classes):  [  9.10912158   2.10785636 -11.21697794]
logistic Regression Accuracy: 100.00%
```

SAVE PREDICTIONS

In [ ]:
```
# Save the predictions for uture use or comparision
# Save predictions to a new Data Frame
```

```
predictions = X_test.copy()
predictions["Actual Price"] = y_test
predictions["Predicted Price"] = y_pred_logreg

# Save to a CSV file
predictions.to_csv("iris_dataset_predictions.csv",index=False)
print("Predictions saved to 'iris_dataset_predictions.csv' ")
```

```
Predictions saved to 'iris_dataset_predictions.csv'
```

TRAIN A SUPPORT VECTOR MODEL

In [ ]:

```
from sklearn.svm import SVC

# Create thr SVM model with a linear kernel
svm_model = SVC(kernel='linear')

# Train the model on the training data
svm_model.fit(X_train,y_train)

# make prediction using the test data
y_pred_svm = svm_model.predict(X_test)

# Calculate accuracy of the model
accuracy_svm = accuracy_score(y_test,y_pred_svm)
print(f'Support Vector Machine Accuracy :{accuracy_svm * 100:.2f}%')
```

```
Support Vector Machine Accuracy :100.00%
```

SAVED ENHANCED DATASET

In [ ]:

```
df["Predic Price(logreg)"] = svm_model.predict(X)

# Save to a new CSV
df.to_csv("enhanced_iris_dataset.csv",index=False)
print("enhanced dataset saved as 'enhanced_iris_dataset.csv' ")
```

```
enhanced dataset saved as 'enhanced_iris_dataset.csv'
```

In [ ]: